EECE5644: Assignment #3

Machine Learning and Pattern Recognition

Name : Nikita Vinod Mandal

NUID : 002826995

Question 1:

In this problem, a multilayer perceptron (MLP) was employed to approximate class label posteriors. The training of the MLP involved minimizing the average cross-entropy loss. The trained models were then used to approximate a Maximum A Posteriori (MAP) classification rule, aiming for minimal probability of error on a validation dataset.

Dataset Generation: A 3-dimensional real-values random vector, denoted as x, was generated to belong to one of four classes with uniform priors and Gaussian class conditional probability density functions (pdfs).

$P(L = l) = 0.2, for\ l = [0,1,2,3]$

Class Conditional Parameters:

$\boldsymbol{m}0$= [1 1 0], $\boldsymbol{C}0$= [1.00 0.01 0.01, 0.01 1.02 0.03, 0.01 0.03 1.06]

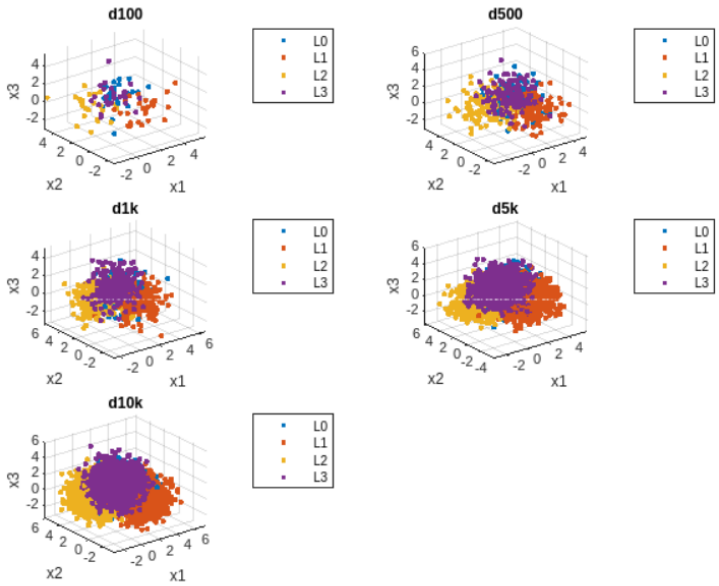
$\boldsymbol{m}1$= [1 0 0], $\boldsymbol{C}1$= [1.05 0.04 0.01, 0.04 1.04 0.01, 0.01 0.01 1]

$\boldsymbol{m}2$= [0 1 0], $\boldsymbol{C}2$=[1.06 0.08 0.05 , 0.06 1.05 0.07 , 0.06 0.05 1.05]

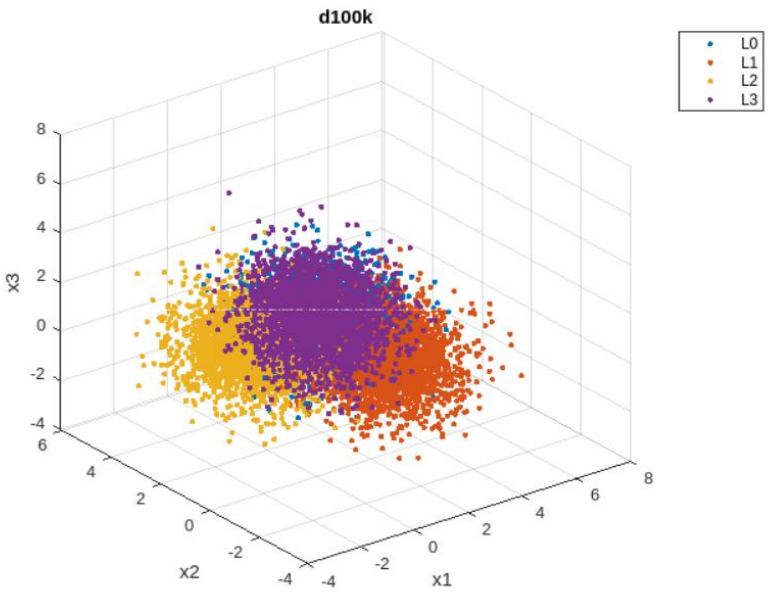
$\boldsymbol{m}3$= [0 0 1], $\boldsymbol{C}3$=[1.03 0.05 0.03 , 0.03 1.05 0.04 ,0.02 0.03 1.02]

MLP Architecture: A 2-layer MLP with one hidden layer and one output layer was specified. The output layer utilized a "softmax" activation function, suitable for multiclass classification. The MATLAB "patternnet" function was employed for implementation, and the number of perceptrons was determined through cross-validation. A smooth-ramp style activation function was activated to mitigate issues related to sigmoid functions that can impede training progress. The Rectified Linear Unit (ReLU) function was employed, providing a half rectified activation from the bottom and ensuring monotonicity.

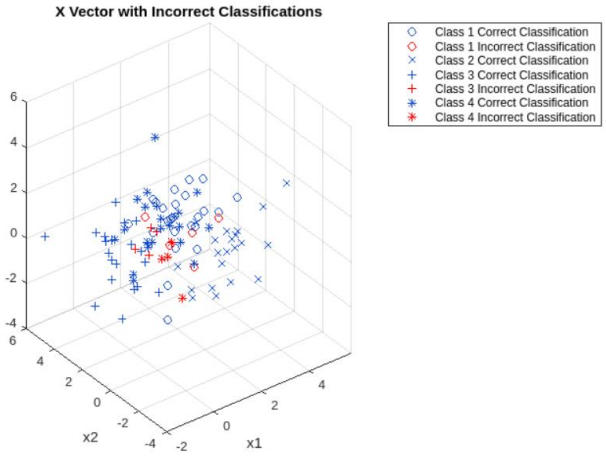
Dataset Sizes: Training datasets with sizes of 100, 500, 1000, 5000, and 10000 samples were generated. Additionally, a validation dataset with 100,000 samples was created for testing the trained models.



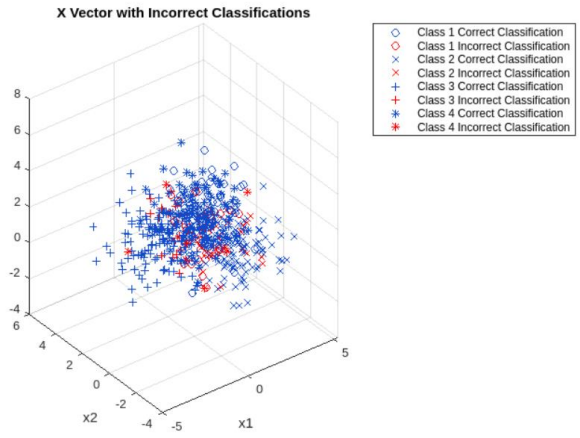
Training Dataset



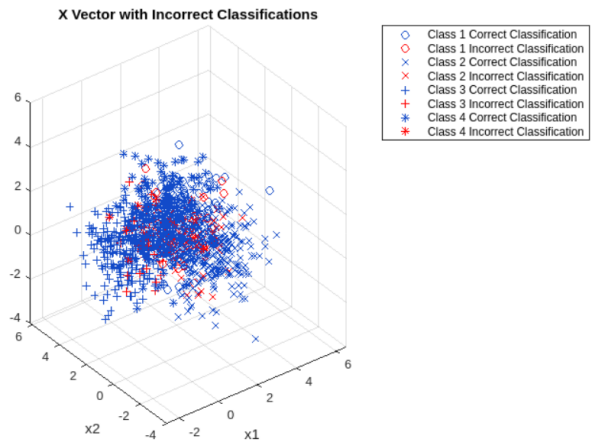
Validation Dataset of 100k Samples



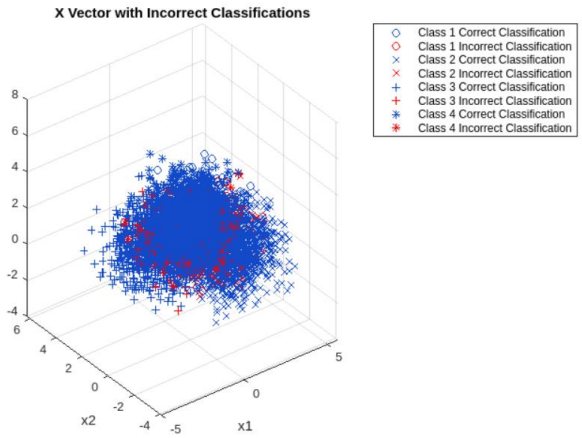
Classification of 100 Sample



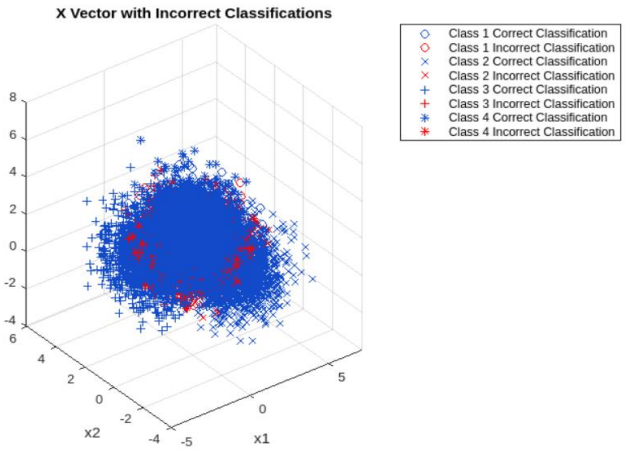
Classification of 500 Samples



Classification of 1000 Samples



Classification of 5000 Samples

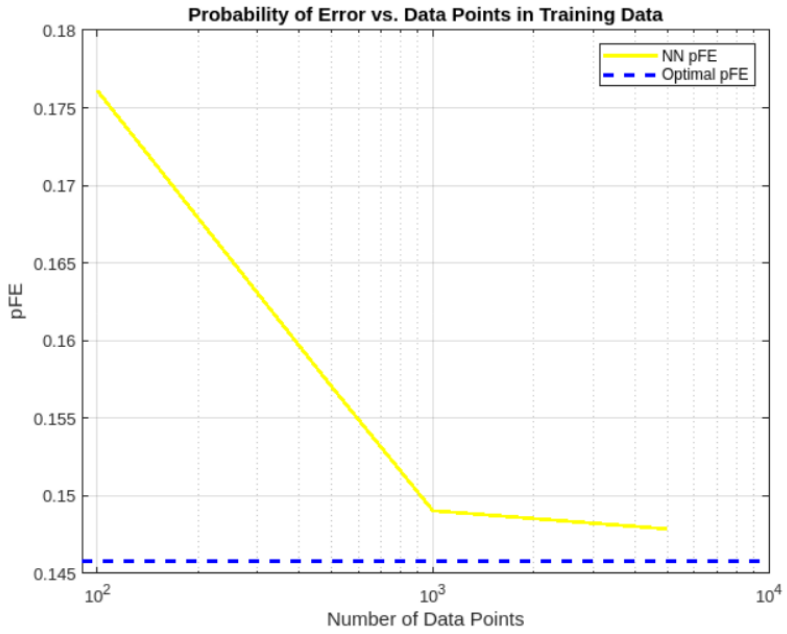


Classification of 10000 Samples

For each training dataset, a 10-fold cross-validation was executed to determine the optimal number of perceptrons for the MLP model. In this process, the fitting procedure was repeated ten times, where each fit was performed on a training set comprising 90% of the total training data selected randomly, and the remaining 10% served as a holdout set for validation. The optimal number of perceptrons was identified based on the minimum probability of error across these cross-validation runs. Once determined, a final model was trained using the entire training dataset.

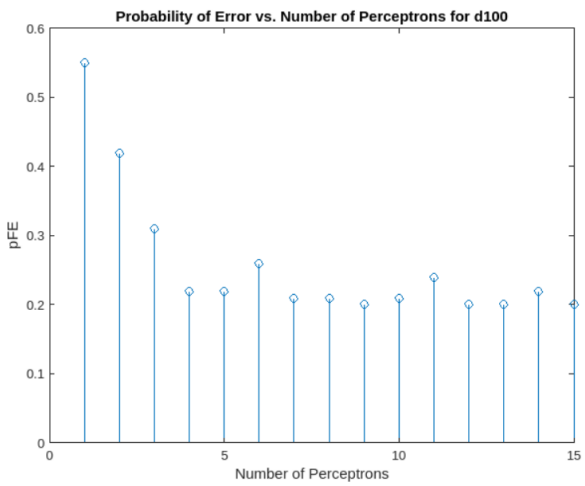
Subsequently, this trained model underwent evaluation using the test dataset, and the probability of error was computed as the performance metric for the model. The presented figure illustrates the outcomes of

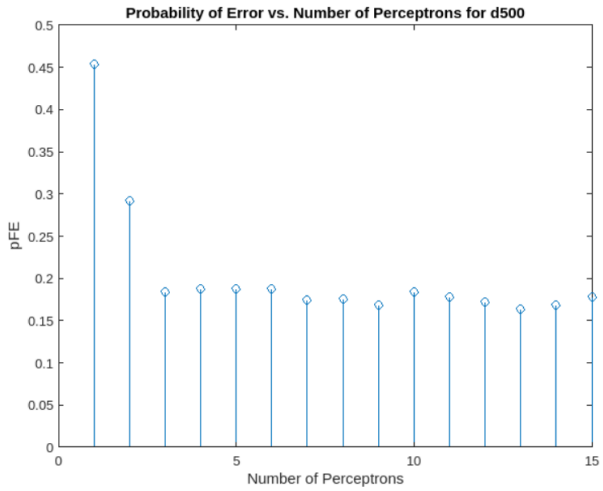
this comprehensive procedure. Notably, the overall probability of error exhibits a strong correlation with the size of the training dataset. As the dataset size increases, the probability of error decreases and converges towards the optimal probability of error, as estimated using the true probability density function of the underlying data. This observation underscores that an augmentation in the quantity of training data leads to improved model estimates, resulting in more accurate classifications.

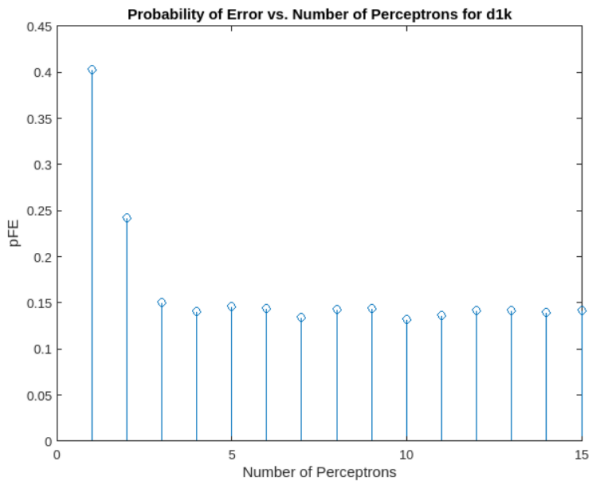


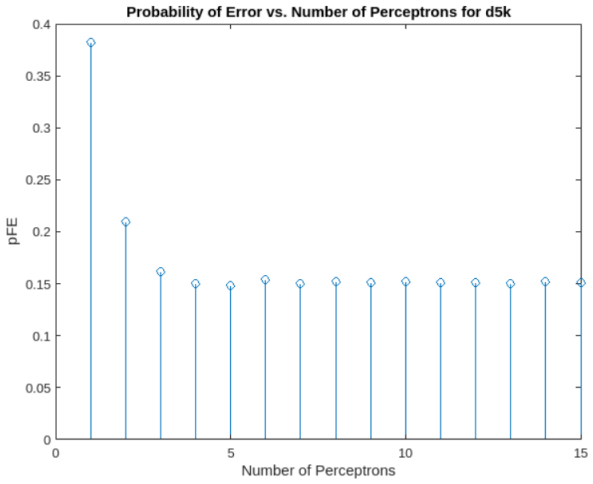
Probability of Error vs. Number of Data Points

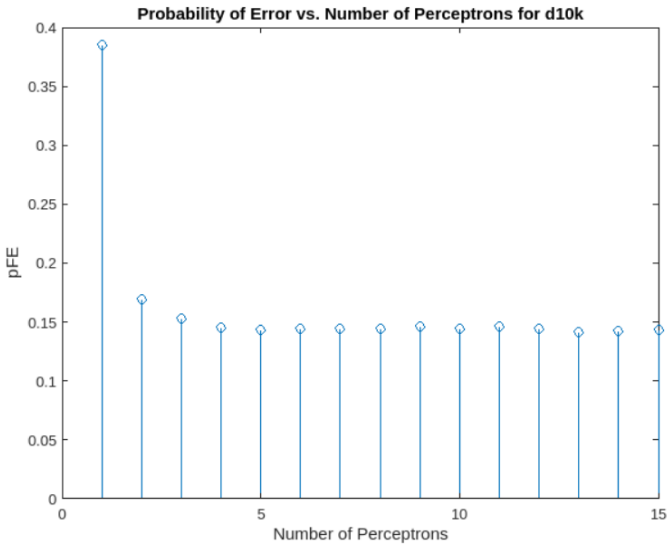
Probability of Error for various test datasets:



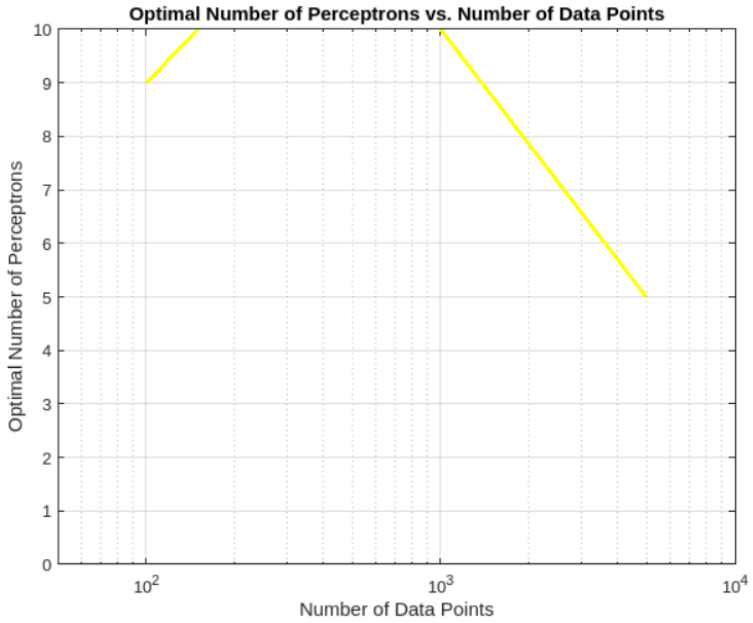








The plot below depicts the optimal number of perceptrons in relation to the number of data points in a dataset. With the exception of the data point for the 1000-point training set, there is a discernible trend where the optimal number of perceptrons tends to increase as the size of the training dataset increases. This outcome aligns with expectations, as a larger dataset allows for a more complex model. The increased data size introduces additional features that the model can capture, leading to a rise in model complexity. However, it is crucial to note that excessively complex models, especially when trained on large datasets, may inadvertently incorporate noise and generate inaccurate probabilities. To mitigate this, the model training process is designed to avoid overfitting, ensuring that it does not capture noise or produce inappropriate probability estimates.



Probability errors are obtained as below: They lie in the range of 10-20%. The gaussian mixtures is selected in such a way that it lies between the 10-20%.

```
Optimal pFE, N=100: Error=13.00%
Optimal pFE, N=500: Error=16.00%
Optimal pFE, N=1000: Error=13.00%
Optimal pFE, N=5000: Error=15.16%
Optimal pFE, N=10000: Error=14.33%
Optimal pFE, N=100000: Error=14.58%
NN pFE, N=100: Error=17.61%
NN pFE, N=500: Error=15.70%
NN pFE, N=1000: Error=14.90%
NN pFE, N=5000: Error=14.79%
NN pFE, N=10000: Error=14.70%
```

As observed the error probability decreases with the increase in the number of samples. NN is the practically obtained probability errors.

Question 2:

The given question involves utilizing a Gaussian Mixture Model (GMM) as the true probability density function for synthesizing 2-dimensional real-valued data. This GMM comprises four components, each characterized by distinct mean vectors, covariance matrices, and probabilities for being selected as the generator for individual samples.

The mean vectors are specified as [-10 10 10 -10; 10 10 -10 -10], indicating the positions of the Gaussian components. Additionally, the covariance matrices for the four components are defined as follows:

Covariance 1: [20 1; 10 3]

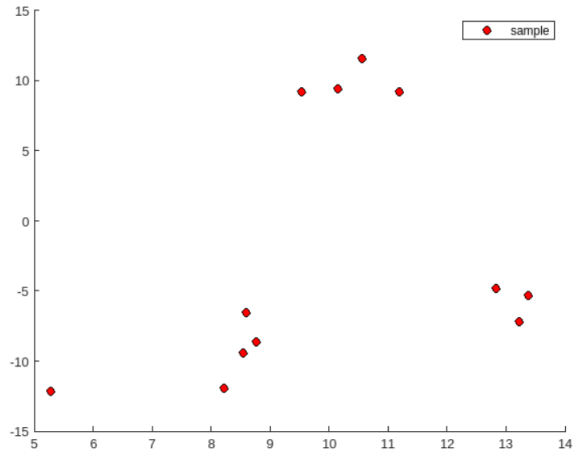
Covariance 2: [7 1; 1 2]

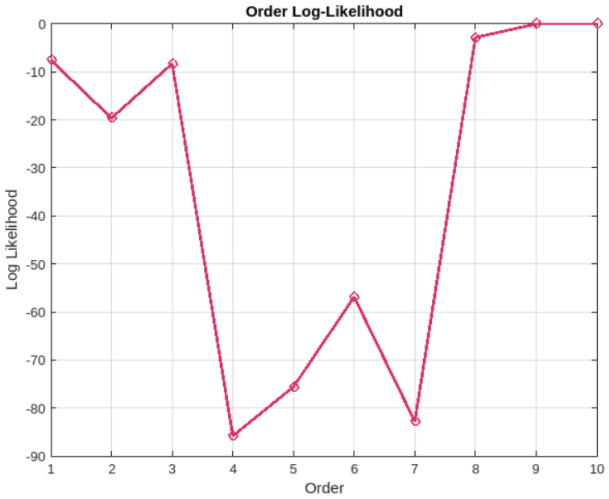
Covariance 3: [4 10; 1 16]

Covariance 4: [2 1; 1 7]

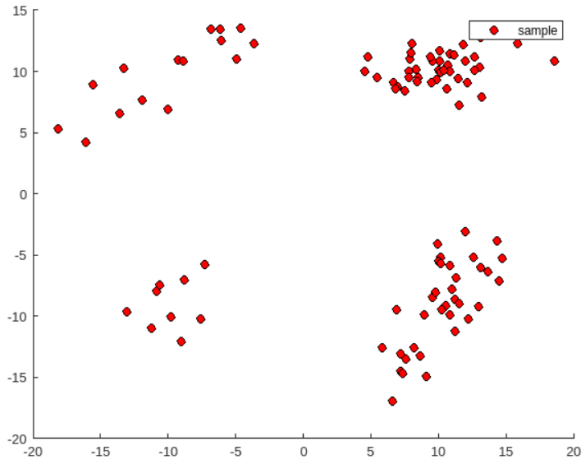
In this configuration, the Gaussian components are deliberately chosen to have a larger separation, emphasizing distinct clusters in the data synthesis. The K-means algorithm is then applied, defining groups during its execution. Once this clustering is established, any new data can be easily assigned to the correct group based on the defined clusters. The resulting clusters are visually evident in each GMM distribution of the generated samples.

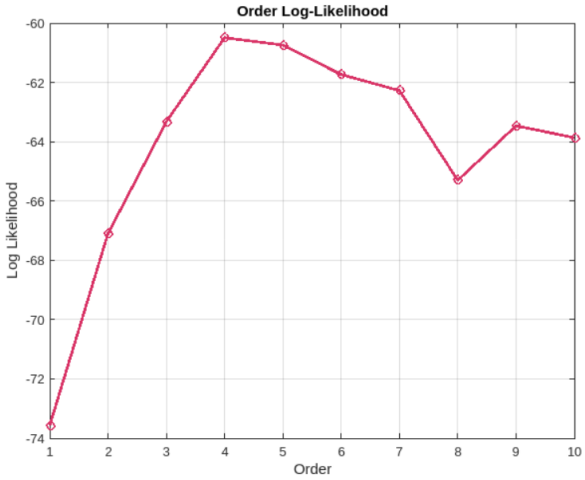
For N=12 the GMM data generated as below.



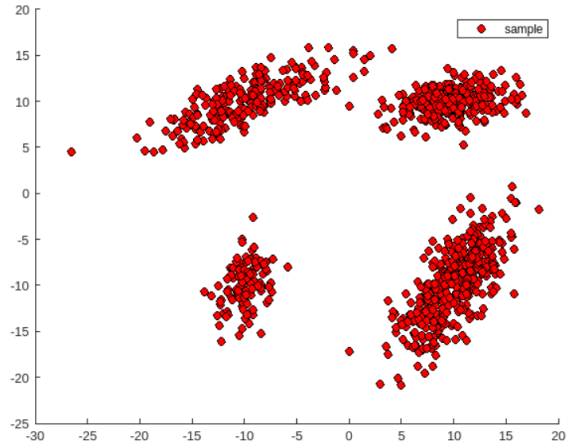


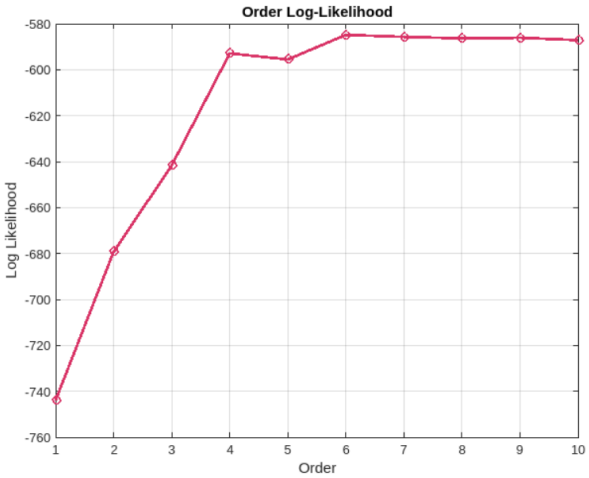
For N=100





For N=1000



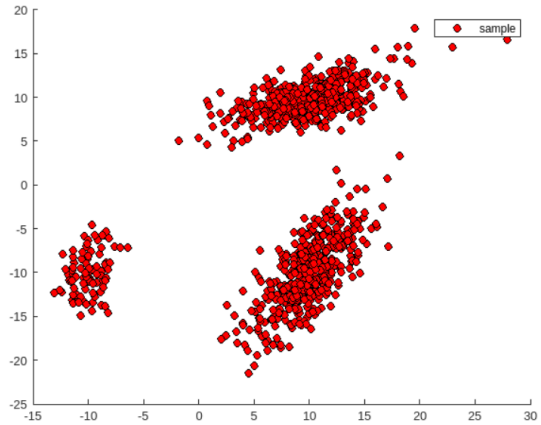


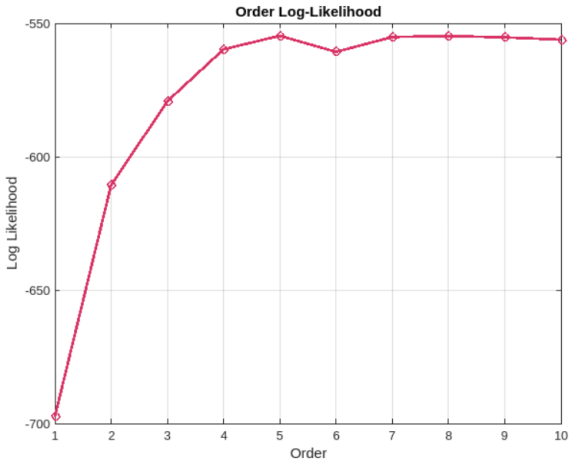
For N=1000

mu_true = [10 10 10 -10;10 10 -10 -10];

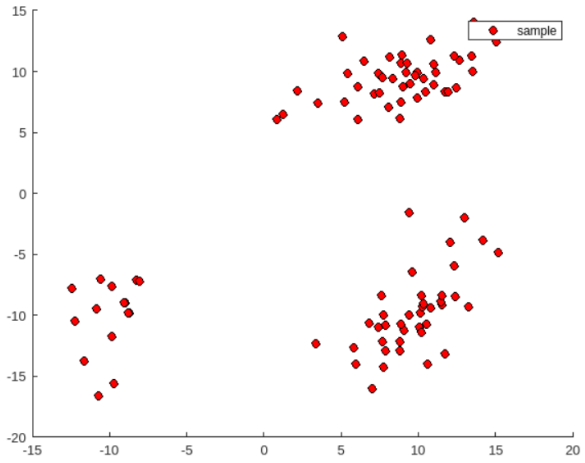
This is the true GMM that generates data in a way that the two Gaussian components are overlapping significantly.

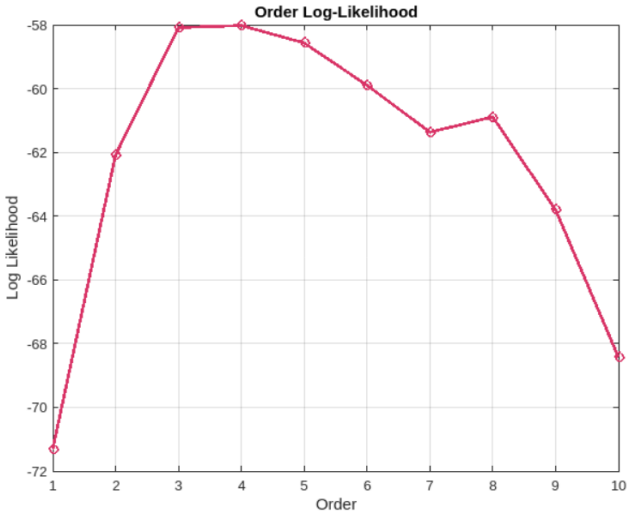
As we can observe from the plot below, the two distinct Gaussian components have now overlapped for the given values for the same data set.



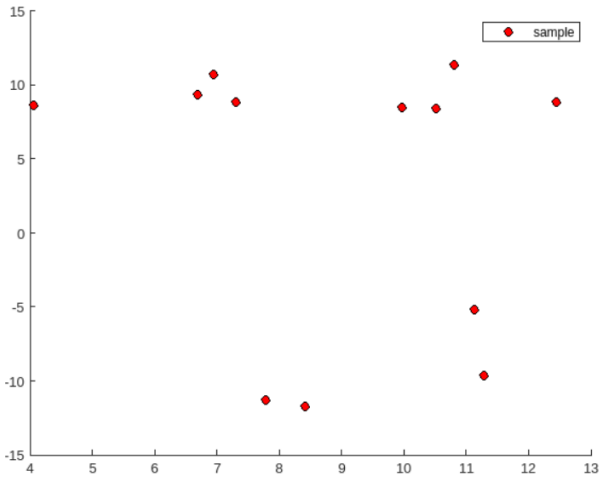


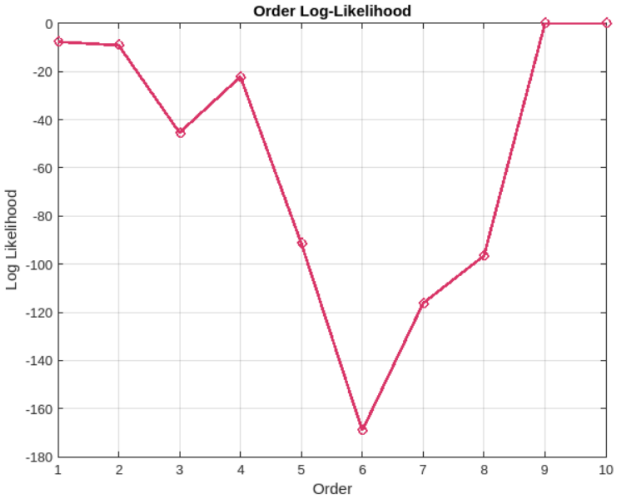
For N=100, mu_true = [10 10 10 -10;10 10 -10 -10];





For N=10, mu_true = [10 10 10 -10;10 10 -10 -10];





Conclusion:

In conclusion, the log likelihood suggests that a model with four components is appropriate for this dataset. Beyond four components, there is no significant improvement, and the graph plateaus. The plateauing occurs rapidly around the fourth component, indicating that it is the most suitable for this Gaussian Mixture Model.

Upon closer observation, for sample sizes ranging from 10 to 100, the graph does not plateau, indicating that a greater number of components is being considered. However, from 1000 samples onwards, the graph starts to converge and remains relatively constant after the sixth component. This trend signifies that as the sample size increases, the model tends to favor a more stable configuration, ultimately converging towards the optimal four-component model.

Appendix :

Question 1:

```
clear all;
close all;

dimensions=3;
numLabels=4;
Lx={'L0', 'L1','L2','L3'};
lossMatrix = ones(numLabels,numLabels)-eye(numLabels);

muScale=2.5;
SigmaScale=0.2;

%Data Defination
D.d100.N=100;
D.d500.N=500;
D.d1k.N=1e3;
D.d5k.N=5e3;
D.d10k.N=10e3;
D.d100k.N=100e3;
dTypes=fieldnames(D);

%Statistics
p=ones(1,numLabels)/numLabels;

mu.L0=muScale*[1 1 0]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L0(:,:,1)=RandSig*RandSig'+eye(dimensions);

mu.L1=muScale*[1 0 0]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L1(:,:,1)=RandSig*RandSig'+eye(dimensions);

mu.L2=muScale*[0 1 0]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L2(:,:,1)=RandSig*RandSig'+eye(dimensions);

mu.L3=muScale*[0 0 1]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L3(:,:,1)=RandSig*RandSig'+eye(dimensions);

mu.L4=muScale*[0 1 1]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L4(:,:,1)=RandSig*RandSig'+eye(dimensions);

%Data Generation

for ind=1:length(dTypes)
    D.(dTypes{ind}).x=zeros(dimensions,D.(dTypes{ind}).N);
    [D.(dTypes{ind}).x,D.(dTypes{ind}).labels,...
        D.(dTypes{ind}).N_l,D.(dTypes{ind}).p_hat]=...
        genData(D.(dTypes{ind}).N,p,mu,Sigma,Lx,dimensions);
end
```

```
%Plot Training Data
figure;
for ind=1:length(dTypes)-1
    subplot(3,2,ind);
    plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);
    legend 'show';
    title([dTypes{ind}]);
end

%Plot Validation Data
figure;
plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);
legend 'show';
title([dTypes{end}]);

%Determine Theoretically Optimal Classifier

for ind=1:length(dTypes)
    [D.(dTypes{ind}).opt.PFE, D.(dTypes{ind}).opt.decisions]=...
        optClass(lossMatrix,D.(dTypes{ind}).x,mu,Sigma,...
        p,D.(dTypes{ind}).labels,Lx);
    opPFE(ind)=D.(dTypes{ind}).opt.PFE;
    fprintf('Optimal pFE, N=%1.0f: Error=%1.2f%%\n',...
    D.(dTypes{ind}).N,100*D.(dTypes{ind}).opt.PFE);
end

%Train and Validate Data

numPerc=15; %Max number of perceptrons to attempt to train
k=10; %number of folds for kfold validation
for ind=1:length(dTypes)-1
    %kfold validation is in this function
    [D.(dTypes{ind}).net,D.(dTypes{ind}).minPFE,...
        D.(dTypes{ind}).optM,valData.(dTypes{ind}).stats]=...
        kfoldMLP_NN(numPerc,k,D.(dTypes{ind}).x,...
        D.(dTypes{ind}).labels,numLabels);

    %Produce validation data from test dataset
    valData.(dTypes{ind}).yVal=D.(dTypes{ind}).net(D.d100k.x);

    [~,valData.(dTypes{ind}).decisions]=max(valData.(dTypes{ind}).yVal);
    valData.(dTypes{ind}).decisions=valData.(dTypes{ind}).decisions-1;

    %Probability of Error is wrong decisions/num data points
    valData.(dTypes{ind}).pFE=...
        sum(valData.(dTypes{ind}).decisions~=D.d100k.labels)/D.d100k.N;
    outpFE(ind,1)=D.(dTypes{ind}).N;
    outpFE(ind,2)=valData.(dTypes{ind}).pFE;
    outpFE(ind,3)=D.(dTypes{ind}).optM;

    fprintf('NN pFE, N=%1.0f: Error=%1.2f%%\n',...
        D.(dTypes{ind}).N,100*valData.(dTypes{ind}).pFE);
end
```

```
for ind=1:length(dTypes)-1
    [~,select]=min(valData.(dTypes{ind}).stats.mPFE);
    M(ind)=(valData.(dTypes{ind}).stats.M(select));
    N(ind)=D.(dTypes{ind}).N;
end

%Plot number of perceptrons vs. pFE for the cross validation runs
for ind=1:length(dTypes)-1
    figure;
    stem(valData.(dTypes{ind}).stats.M,valData.(dTypes{ind}).stats.mPFE);
    xlabel('Number of Perceptrons');
    ylabel('pFE');
    title(['Probability of Error vs. Number of Perceptrons for ' dTypes{ind}]);
end



%Number of perceptrons vs. size of training dataset
figure,semilogx(N(1:end-1),M(1:end-1),'y','LineWidth',2)
grid on;
xlabel('Number of Data Points')
ylabel('Optimal Number of Perceptrons')
ylim([0 10]);
xlim([50 10^4]);
title('Optimal Number of Perceptrons vs. Number of Data Points');

%Prob. of Error vs. size of training data set
figure,semilogx(outpFE(1:end-1,1),outpFE(1:end-1,2),'y','LineWidth',2)
xlim([90 10^4]);
hold all;semilogx(xlim,[opPFE(end) opPFE(end)],'b--','LineWidth',2)
legend('NN pFE','Optimal pFE')
grid on
xlabel('Number of Data Points')
ylabel('pFE')
title('Probability of Error vs. Data Points in Training Data');

%Function Definitions

function [x,labels,N_l,p_hat]= genData(N,p,mu,Sigma,Lx,d)
%Generates data and labels for random variable x from multiple gaussian distributions
numD = length(Lx);
cum_p = [0,cumsum(p)];

u = rand(1,N);
x = zeros(d,N);
labels = zeros(1,N);

for ind=1:numD
    pts = find(cum_p(ind)<u & u<=cum_p(ind+1));
    N_l(ind)=length(pts);
    x(:,pts) = mvnrnd(mu.(Lx{ind}),Sigma.(Lx{ind}),N_l(ind))';
    labels(pts)=ind-1;
    p_hat(ind)=N_l(ind)/N;
end
end
```

```
function plotData(x,labels,Lx)
%Plots data
for ind=1:length(Lx)
    pindex=labels==ind-1;
    plot3(x(1,pindex),x(2,pindex),x(3,pindex),'.','DisplayName',Lx{ind});
    hold all;
end
grid on;
xlabel('x1');
ylabel('x2');
zlabel('x3');

end

function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(invSigma*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end

function [outputNet,outputPFE, optM, stats]=kfoldMLP_NN(numPerc,k,x,labels,numLabels)
%Assumes data is evenly divisible by partition choice which it should be

N=length(x);
numValIters=10;

%Create output matrices from labels
y=zeros(numLabels,length(x));
for ind=1:numLabels
    y(ind,:)=(labels==ind-1);
end

%Setup cross validation on training data
partSize=N/k;
partInd=[1:partSize:N length(x)];

%Perform cross validation to select number of perceptrons
for M=1:numPerc
    for ind=1:k
        index.val=partInd(ind):partInd(ind+1);
        index.train=setdiff(1:N,index.val);

        %Create object with M perceptrons in hidden layer
        net=patternnet(M);
            %Train using training data
        net=train(net,x(:,index.train),y(:,index.train));
        %Validate with remaining data
        yVal=net(x(:,index.val));

        [~,labelVal]=max(yVal);
        labelVal=labelVal-1;
```

```
        pFE(ind)=sum(labelVal~=labels(index.val))/partSize;
    end

    %Determine average probability of error for a number of perceptrons
    avgPFE(M)=mean(pFE);
    stats.M=1:M;
    stats.mPFE=avgPFE;
end

%Determine optimal number of perceptrons
[~,optM]=min(avgPFE);

%Train one final time on all the data
for ind=1:numValIters
    netName(ind)={['net' num2str(ind)]};
    finalnet.(netName{ind})=patternnet(optM);
        % finalnet.layers{1}.transferFcn = 'softplus';%Set to RELU
    finalnet.(netName{ind})=train(net,x,y);
    yVal=finalnet.(netName{ind})(x);
    [~,labelVal]=max(yVal);
    labelVal=labelVal-1;

    pFEFinal(ind)=sum(labelVal~=labels)/length(x);
end

[minPFE,outInd]=min(pFEFinal);
stats.finalPFE=pFEFinal;

outputPFE=minPFE;
outputNet=finalnet.(netName{outInd});
end

function [minPFE,decisions]=optClass(lossMatrix,x,mu,Sigma,p,labels,Lx)
% Determine optimal probability of error
symbols='ox+*v';
numLabels=length(Lx);
N=length(x);

for ind = 1:numLabels
    pxgivenl(ind,:) =...
        evalGaussian(x,mu.(Lx{ind}),Sigma.(Lx{ind})); % Evaluate p(x|L=l)
end

px = p*pxgivenl; % Total probability theorem
classPosteriors = pxgivenl.*repmat(p',1,N)./repmat(px,numLabels,1);
%P(L=l|x)

% Expected Risk for each label (rows) for each sample (columns)
expectedRisks =lossMatrix*classPosteriors;

% Minimum expected risk decision with 0-1 loss is the same as MAP
[~,decisions] = min(expectedRisks,[],1);
decisions=decisions-1; %Adjust to account for L0 label
```

```
fDecision_ind=(decisions~=labels);%Incorrect classificiation vector

minPFE=sum(fDecision_ind)/N;
%Plot Decisions with Incorrect Results
figure;
for ind=1:numLabels
    class_ind=decisions==ind-1;
    plot3(x(1,class_ind & ~fDecision_ind),...
        x(2,class_ind & ~fDecision_ind),...
        x(3,class_ind & ~fDecision_ind),...
        symbols(ind),'Color',[0.07 0.29 0.79],'DisplayName',...
        ['Class ' num2str(ind) ' Correct Classification']);
    hold on;
    plot3(x(1,class_ind & fDecision_ind),...
        x(2,class_ind & fDecision_ind),...
        x(3,class_ind & fDecision_ind),...
        ['r' symbols(ind)],'DisplayName',...
        ['Class ' num2str(ind) ' Incorrect Classification']);
    hold on;
end

xlabel('x1');
ylabel('x2');n
grid on;

title('X Vector with Incorrect Classifications');
legend 'show';

if 0
    %Plot Decisions with Incorrect Decisions
    figure;
    for ind2 = 1:numLabels
        subplot(3,2,ind2);
        for ind = 1:numLabels
            class_ind = decisions == ind-1;
            plot3(x(1, class_ind), x(2, class_ind), x(3, class_ind), '.',
'DisplayName', ['Class ' num2str(ind)]);
            hold on;
        end
        plot3(x(1, fDecision_ind & labels == ind2), x(2, fDecision_ind & labels ==
ind2), x(3, fDecision_ind & labels == ind2), 'kx', 'DisplayName', 'Incorrectly
Classified', 'LineWidth', 2);
        ylabel('x2');
        grid on;
        title(['X Vector with Incorrect Decisions for Class ' num2str(ind2)]);

        if ind2 == 1
            legend('show');
        elseif ind2 == 4
            xlabel('x1');
        end
    end

end
end
```

Question 2:

```
clear;
close all;

% taking the values and trying each time with the N as
% 10,100,1000
N = 1000;
delta = 1e0; % tolerance for EM stopping criterion
regWeight = 1e-10; % regularization parameter for covariance estimates
% Replicating it 30 times
% Generate samples from a 4-component GMM
alpha_true = [0.2,0.3,0.4,0.1];
mu_true = [-10 10 10 -10;10 10 -10 -10];
Sigma_true(:,:,1) = [20 1;10 3];
Sigma_true(:,:,2) = [7 1;1 2];
Sigma_true(:,:,3) = [4 10;1 16];
Sigma_true(:,:,4) = [2 1;1 7];
x = randomGMM(N,alpha_true,mu_true,Sigma_true);

figure(1);
figure(1), scatter(x(1,:), x(2,:), 'Marker', 'o', 'MarkerFaceColor', 'red', 'MarkerEdgeColor', 'k'), hold on,
figure(1),legend('sample')
d = 2;
K = 10;
dummy = ceil(linspace(0,N,K+1));
for k = 1:K
   indPartitionLimits(k,:) = [dummy(k)+1,dummy(k+1)];
end

avgp = zeros(1, min(10, N));

for M = 1:min(10, N)
   psum = zeros(1, 10);
   for k = 1:K
      indValidate = [indPartitionLimits(k,1):indPartitionLimits(k,2)];
      xValidate = x(:,indValidate);
      if k == 1
         indTrain = [indPartitionLimits(k,2)+1:N];
      elseif k == K
         indTrain = [1:indPartitionLimits(k,1)-1];
      else
         indTrain = [[1:indPartitionLimits(k-1,2)],[indPartitionLimits(k+1,1):N]];
      end
      xTrain = x(:,indTrain);
      Ntrain = length(indTrain);
      Nvalidate = length(indValidate);
      [alpha,mu,Sigma] = EMforGMM(Ntrain, xTrain, M, d, delta, regWeight);

      % determine the dimensionality of samples and the number of GMM components
      p = zeros(1, Nvalidate);
      for j = 1:Nvalidate
         for i = 1:M
            p(j) = p(j) + alpha(i)*evaluateGauss(xValidate(:,j), mu(:,i), Sigma(:,:,i));
         end
         p(j) = log(p(j));
```

```
            % Check if log likelihood is not -inf or NaN
            if isfinite(p(j))
                psum(k) = psum(k) + p(j);
            end
        end
    end

    % Calculate the average log likelihood excluding -inf and NaN
    avgp(M) = nansum(psum) / sum(isfinite(psum));
end

% Display the order log-likelihood graph
figure(2);
plot(1:min(10, N), avgp, 'o-', 'LineWidth', 2, 'Color', [0.85, 0.2, 0.4]);
xlabel('Order');
ylabel('Log Likelihood');
title('Order Log-Likelihood');
grid on;

function [alpha, mu, Sigma] = EMforGMM(N, x, M, d, delta, regWeight)
    % Initialize the GMM to randomly selected samples
    alpha = ones(1, M) / M;
    mu = datasample(x, M, 2, 'Replace', false); % Adjusted this line
    [~, assignedCentroidLabels] = min(pdist2(mu', x'), [], 1); % assign each sample to the nearest mean
    for m = 1:M
        % use sample covariances of initial assignments as initial covariance estimates
        Sigma(:, :, m) = cov(x(:, assignedCentroidLabels == m)') + regWeight * eye(d, d);
    end
    t = 0;

    Converged = 0;
    while ~Converged
        temp = zeros(M, N);
        for l = 1:M
            temp(l, :) = alpha(l) * evaluateGauss(x, mu(:, l), Sigma(:, :, l));
        end
        plgivenx = temp ./ sum(temp, 1);
        alphaNew = mean(plgivenx, 2);
        w = plgivenx ./ repmat(sum(plgivenx, 2), 1, N);
        muNew = x * w';
        for l = 1:M
            v = x - repmat(muNew(:, l), 1, N);
            u = repmat(w(l, :), d, 1) .* v;
            SigmaNew(:, :, l) = u * v' + regWeight * eye(d, d);
        end
        Dalpha = sum(abs(alphaNew - alpha'));
        Dmu = sum(sum(abs(muNew - mu)));
        DSigma = sum(sum(abs(abs(SigmaNew - Sigma))));
        Converged = ((Dalpha + Dmu + DSigma) < delta);
        alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
        t = t + 1;
    end
end

function x = randomGMM(N,alpha,mu,Sigma)
    d = size(mu,1); % dimensionality of samples
```

```
    cum_alpha = [0,cumsum(alpha)];
    u = rand(1,N); x = zeros(d,N); labels = zeros(1,N);
    for m = 1:length(alpha)
        ind = find(cum_alpha(m)<u & u<=cum_alpha(m+1));
        x(:,ind) = randomGauss(length(ind),mu(:,m),Sigma(:,:,m));
    end
end
%%%
function x = randomGauss(N,mu,Sigma)
    % Generates N samples from a Gaussian pdf with mean mu covariance Sigma
    n = length(mu);
    z =  randn(n,N);
    A = Sigma^(1/2);
    x = A*z + repmat(mu,1,N);
end
function g = evaluateGauss(x,mu,Sigma)
    % Evaluates the Gaussian pdf N(mu,Sigma) at each column of X
    [n,N] = size(x);
    invSigma = inv(Sigma);
    C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
    E = -0.5 * sum((x - repmat(mu, 1, N)) .* (invSigma * (x - repmat(mu, 1, N))), 1);
    g = C * exp(E);
end
```

Citations:1. Credits to Prof. Erdogmus Deniz Notes
2. https://github.com/